DEVELOPMENT OF A SELF-POWERED SENSOR, STORAGE AND TRANSMISSION SYSTEM FOR STRUCTURAL HEALTH MONITORING

A Thesis

by

AALHAD RAJAN PARULEKAR

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Aydin Ilker Karsilayan
Committee Members,	Jose Silva Martinez
	Laszlo Kish
	Michael Pate
Head of Department,	Miroslav Begovic

August 2018

Major Subject: Electrical Engineering

ABSTRACT

A self-powered sensor, storage and transmission system was developed for Structural Health Monitoring (SHM) of transportation infrastructure to mitigate breakdown of roads and provide insights on the health of civil infrastructure prior to actual breakdown. The proposed system, to be embedded under the surface of the road, is designed to power up and enable wireless transmission in a transceiver.

The source of power is thermo-electric, which eliminates the need for batteries, keeping in mind the limitations of battery life in the context of the life-cycle of roads, the latter having a much longer lifespan. The voltage from the Thermoelectric Generator (TEG) is boosted and subsequently regulated to a usable voltage for the proposed application. Vibrations from oncoming cars on the road are converted to an analog voltage by a piezo-electric device. This is fed to the MSP430F2274 micro-controller, which is a part of the End Device (ED), which also has a wireless CC2500 transceiver embedded on-board. The micro-controller has an ADC (analog to digital converter) which digitizes the voltage. This is subsequently transmitted wirelessly by the transceiver to a receiver on an Access Point (AP) when the latter is in range.

The transceiver/controller board has a memory of 32 kB, which can be enhanced using external memory. This will be used for storing oncoming data until an Access Point device is in range, at which point, the digitized data is transmitted. The array of data received at the Access Point is analyzed to collect statistical information including number of vehicles (type, weight etc.) and further studied for insights into breakdown of the road.

DEDICATION

To Aai and Baba – my eternal guiding lights.

ACKNOWLEDGMENTS

I would like to thank Dr.Aydin Karsilayan, for his continuous guidance and support during the course of this thesis.

I want to thank Dr. Samer Dessouky and Dr. Athanassios T. Papagiannakis of the University of Texas at San Antonio for their insights and support.

Thanks are also due to my roommates - Sudhanshu, Ishan and Shubham for having my back through thick and thin. Special thanks to Siddarth Biyani for all the help in the lab and for being a constant source of ideas.

Lastly and most importantly, my parents without whom none of this would have been possible.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of Dr. Aydin Ilker Karsilayan, Dr. Jose Silva-Martinez and Dr. Laszlo Kish of the Department of Electrical Engineering Department Dr. Michael Pate of the Department of Mechanical Engineering.

The year-long temperature data from roads in Minnesota and San Antonio as well as the setup for the heat-transfer device prototype was provided by the department of Civil Engineering at the University of Texas, San Antonio.

Funding Sources

This project is funded by the Transportation Consortium of South Central States. All other work conducted for the thesis was completed by the student independently.

NOMENCLATURE

SHM	Structural Health Monitoring
TEG	Thermo-Electric Generator
ADC	Analog to Digital Converter
RF	Radio Frequency
DC	Direct Current
BiTe	Bismuth Telluride
AP	Access Point
ED	End Device

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION AND PROBLEM STATEMENT

Post World War II, The Federal Aid Highway Act of 1956 was one of the most predominant factors that ushered the United States into the era of massive economic growth and prosperity. As of 2017, the nation has more than 6.5 million kilometers of roads across its length and breadth, making it the largest road network in the world. However, a network of this size also faces a maintenance problem of an equal magnitude. The economic corridors which opened up due to excellent road connectivity, led to an ever-increasing number of vehicles on American roads. The massive cost of repair for a project of this size notwithstanding, the difficulty of maintenance means that US road infrastructure is crumbling further [11].

Since a majority of roads (especially highways) are in use for a significant portion of the year, closing a portion of the road for repair is often inconvenient, and, at times unfeasible. Therefore, it is crucial to mitigate the dangers of road breakdown before the actual deterioration of the road. This way, repairs on that part of the road will consume lesser effort, lesser cost and cause lesser inconvenience. To this end, this work discusses a Self-powered Structural Health Monitoring System (SHM) with the goal of extending the life of infrastructure systems such as roads and pavements. This solution is especially relevant for rural roads with little access to the electric grid, which make up a majority of Texas roadways.

The need to build a self-powered (battery-less) SHM system stems from the very nature of this project. Not only would having several batteries increase the overall cost of the project, they would also prove to be unfeasible on the long run. Even the battery with the most superior life-cycle would not outlive the durability of a road, thus defeating the purpose of the project. Despite the advantages of present-day rechargeable batteries, loss

in capacity (aging) is inevitable and not restorable. Eventually, lesser and lesser active material is available within each battery to electro-chemically store a charge. We therefore resort to thermal sources to power up the entire system.

There have been several approaches to harvest ambient energy in order to use it to power up SHM systems. Zhou et al. [12] employ the harvesting of piezoelectric vibrations to power up the Microcontroller (MSP430). They, however, use a battery for power backup during low-traffic times (when less piezoelectric vibration takes place). Dondi et al. [13] also use piezoelectric vibrations to harvest energy (with a battery backup), but deliver an output power of 0.845 mW and provide a maximum data rate of only 22 bits per second. Hassan et al. [14] employ Solar power as their source, and generate an output as high as 120 mW. However, they also have to rely on a battery backup in order to ensure continuous power supply. Yuan and Wang [15] use Magnetorestrictive materials to harvest energy, and are able to ensure continuous power supply without a battery backup. However, they are reporting a power output of 0.576 mW which is insufficient for the proposed application.

This work discusses the development and demonstration of a self-powered Thermo-electric Power Harvesting System designed for continuous operation, regardless of weather conditions, and the ability of such a system to reliably transfer data to assist in Structural Health Monitoring.

2. PROPOSED SYSTEM FOR HARVESTING OF THERMAL ENERGY

2.1 System Overview

The unique feature of the proposed system is its independence from batteries, in order to gain a long product life. To this end, the entire system is powered from temperature gradient across the surfaces of Thermo-electric generators (TEGs). It converts a temperature difference across its plates to an output voltage (Seeback effect). For the temperature differences that were obtained in the year-long data used in this work, the TEG gives a voltage output of several hundred millivolts. An illustrative System diagram can be seen in Fig. 2.1.

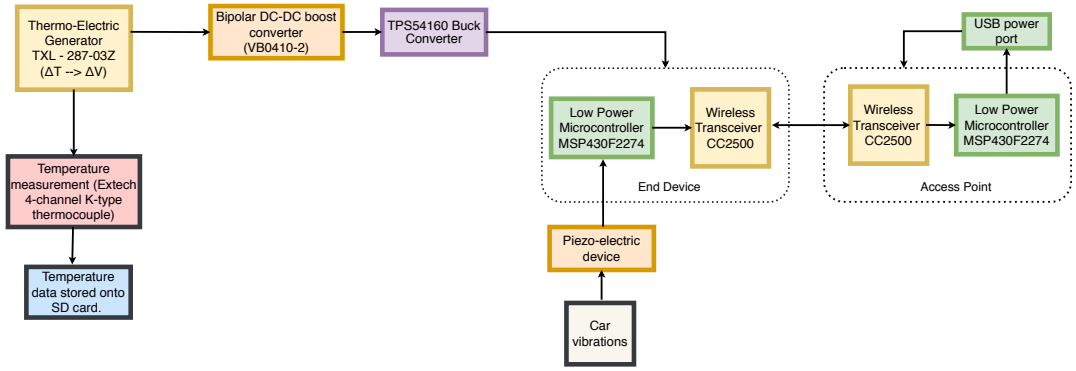

Figure 2.1: System block diagram

The voltage obtained at the output of the TEG (of the order of a few hundred millivolts) is fed to a bipolar DC-DC booster, which boosts it to 9.86 V for input voltages above 40 mV. Typical low-power microcontrollers accept supply voltages much lower than that provided by the DC-DC booster. To this end, the boosted voltage is fed to a Buck converter (TPS54160) in order obtain a lower regulated supply voltage of 3.3 V. This is used to power

up the End Device, which has the MSP430F2274 microcontroller and CC2500 wireless transceiver. The End Device takes an analog input from a piezoelectric device, and stores it in memory after digitizing. The stored data is transmitted wirelessly by the CC2500 Wireless Transceiver to an Access Point (typically located in a remote car/station).

2.2 Thermoelectric Generator

A Thermoelectric generator (TEG) is a device which produces a voltage at its output when a temperature difference is applied across its plates. It works on the principle of the Seeback effect.

Seebeck effect is the production of a voltage in a device when two dissimilar materials form a loop with different temperatures at each junction of the device [16]. When heat is applied to one of the two conducting surfaces, heated electrons, now at a higher potential energy flow towards the cooler one. If such a pair is connected through an electrical circuit, current flows through that circuit [17].

The voltages thu produced by Seebeck effect are of the order of a few microvolts per degree Celsius of junction temperature gradient. Several such devices can be connected in series to increase the output voltage or in parallel to increase the maximum deliverable current. Large arrays of Seebeck-effect devices can provide useful electrical power if a sufficiently high temperature gradient is maintained across the junctions. The voltage produced in a device working on Seeback effect is given by:

$$\Delta V = S * \Delta T \tag{2.1}$$

where S is the Seeback coefficient of the TEG and ΔV is the thermo-electric voltage seen at the terminals, on the application of a temperature difference of ΔT.

2.2.1 Choice Of Thermo-Electric Device

We have chosen a TEG based on Bismuth Telluride (Bi_2Te_3) as the thermoelectric material, which is able to operate at wide temperature ranges from -400^{0}C to 1250^{0}C, and capable of supporting temperature gradients of up to 670^{0}C across its plates. The physical properties of BiTe are a function of its thickness. This enables us to stack several Thermo-electric elements in a module to increase the voltage output per ΔT, from the TEG.

In the proposed work, we have used the TXL-287-03Z TEG from the TXL group. It has 287 BiTe Thermo-electric elements to convert ΔT to voltage [18]. BiTe has a Seebeck coefficient of

$$S = 2X10^{-4}V/^0C \tag{2.2}$$

at room temperature that increases 0.4% per degree Celsius of ΔT. This is expected to give an open circuit output voltage V_{OC} as:

$$V_{OC} = 2 * N * (0.0002 * 1.004^{\Delta T)}) * \Delta T \tag{2.3}$$

where N is the number of thermoelectric elements in the TEG, (in this case 287), and ΔT is the temperature difference between the two plates.

We used a hotplate to heat one plate of the TEG and cooled the other plate using an ice-pack. We then calculated the temperature difference and the corresponding open circuit voltage generated. The TEG used in this work is shown in Fig 2.2 [1]. The open loop voltage given by the TEG vs. the temperature difference across its plates is tabulated in Table 2.1, and plotted graphically in Fig. 2.3.

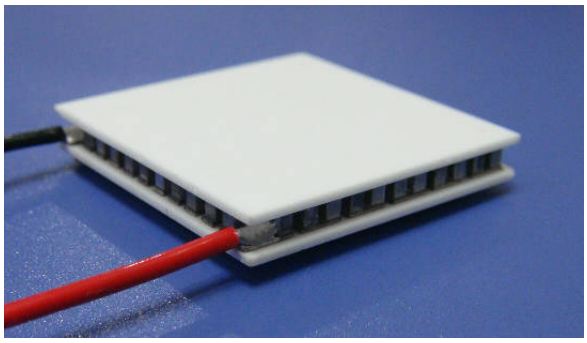

Figure 2.2: Thermoelectric generator (Reprinted from [1])

Table 2.1: TEG open loop voltage given by the TEG vs. ΔT

Plate 1 Temperature (^0C)	Plate 2 Temperature (^0C)	ΔT (^0C)	Measured V_{oc} (mV)	Calculated V_{oc} (mV)
26.2	25.6	0.6	69	69.0
28.3	27.4	0.9	105	103.7
28.6	27.6	1	113	115.2
28.8	27.7	1.1	125	126.8
29.1	27.9	1.2	137	138.4
29.3	28	1.3	150	150.0
30	28.6	1.4	162	161.6
30.5	29	1.5	172	173.2
30.7	29.1	1.6	176	184.8
31.3	29.5	1.8	207	208.1
33.6	31.6	2	227	231.4
33.9	31.8	2.1	241	243.1
34.2	32	2.2	252	254.8
34.4	32.1	2.3	266	266.5

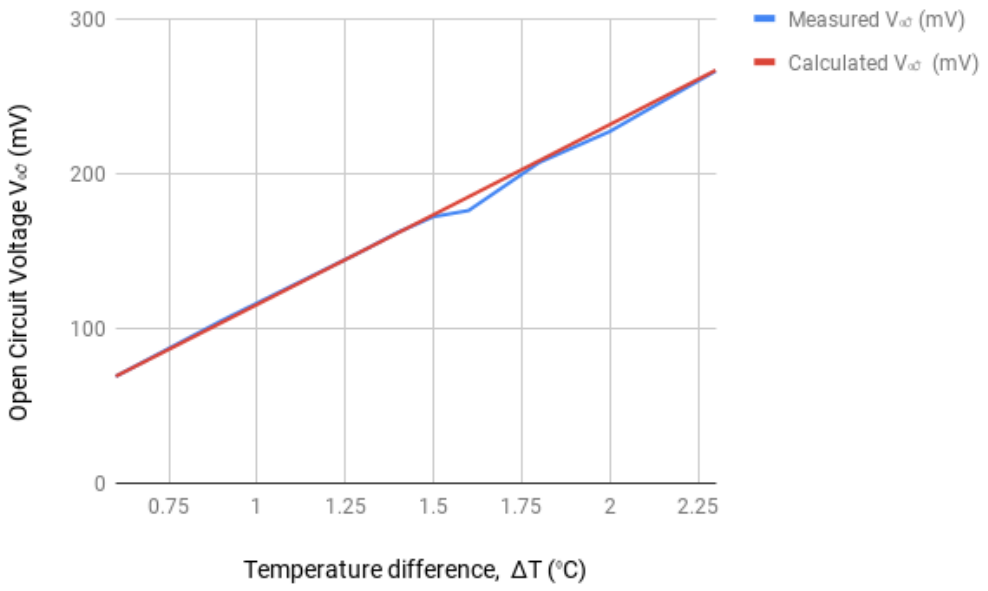

Figure 2.3: Graphical plot of temperature vs. voltage

2.2.2 TEG Setup

As described above, the TEG gives a voltage output for a temperature gradient between its surfaces. This temperature difference is created by the TEG having two different temperatures at its two surfaces (corresponding to two depths) under the road. A copper plate runs from just underneath the surface of the road to the desired depths. This is in contact with the bottom plate of both TEGs. The top plate of both TEGs is in contact with a separate copper plate which is surrounded by underground soil and is thus maintained at the same temperature as that at that particular depth. An illustrative picture can be seen in Figure 2.4.

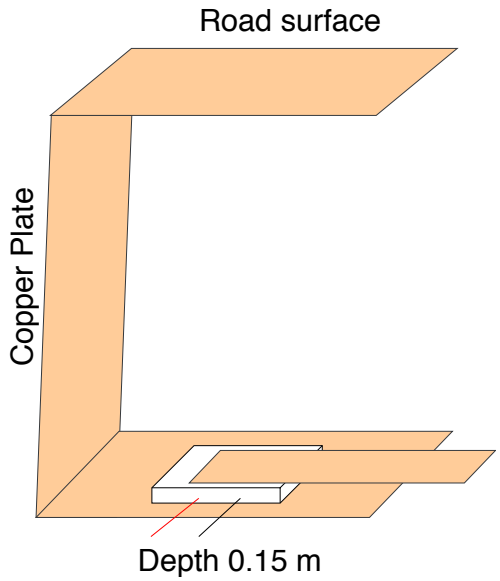

Figure 2.4: Temperature difference maintained by copper plates at 0.15m below the road surface

2.3 DC-DC Booster

The voltage generated from the TEG is in the order of a few hundred millivolts. Therefore, a DC-DC converter is needed to boost the TEG-generated voltage to a level sufficient to power-up the End Device. The DC-DC Booster is classified as a switching regulator. An illustrative diagram of the DC-DC booster is seen in Fig. 2.5 [2].

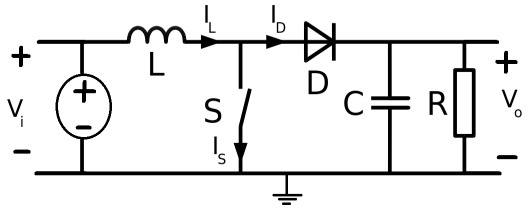

Figure 2.5: DC Boost converter working diagram (Reprinted from [2])

In the proposed system, we use the VB-0410-2 (gold), shown in Fig 2.6 [3], which is a

self-powered bipolar DC-DC converter, giving a positive voltage output regardless of the polarity of the input. This is necessary because the polarity of the temperature gradient between plates varies from night to day, and the bipolar DC-DC converter provides a positive output voltage for positive as well as negative temperature gradients.

This DC-DC booster has a threshold voltage of 40 mV. For inputs exceeding 40 mV, it gives an output of 9.86 V. The boosted outputs of the DC-DC converter vs. the input voltage (obtained from the TEGs) are tabulated in Table 2.2 and the are graphically plotted in Fig 2.7. The output voltage of the booster as a function of the input voltage is given by:

$$V_{out} = \begin{cases} 0V & if \quad \mid V_{in} \mid < 40mV \\ 9.86V & if \quad \mid V_{in} \mid > 40mV \end{cases} \tag{2.4}$$

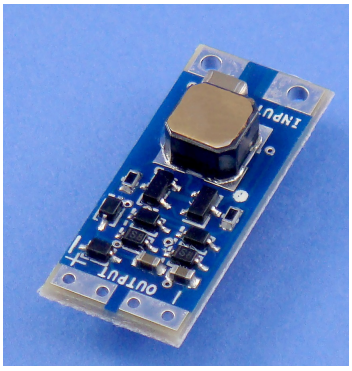

Figure 2.6: VB-0410-2: Bipolar DC-DC booster (Reprinted from [3])

Table 2.2: Input vs. Measured and Ideal DC-DC Booster output voltages

Input Voltage, V_{in} (mV)	Measured Output voltage V_{out} (V)	Ideal output voltage V_{out} (V)
35	0.016	0
36	0.023	0
37	0.048	0
38	0.056	0
39	0.064	0
39.5	0.112	0
39.6	0.356	0
39.7	0.580	0
39.8	0.690	0
39.9	1.2	0
40	3.2	9.86
40.1	6.85	9.86
40.3	7.66	9.86
40.5	8.98	9.86
40.7	9.83	9.86
40.9	9.86	9.86

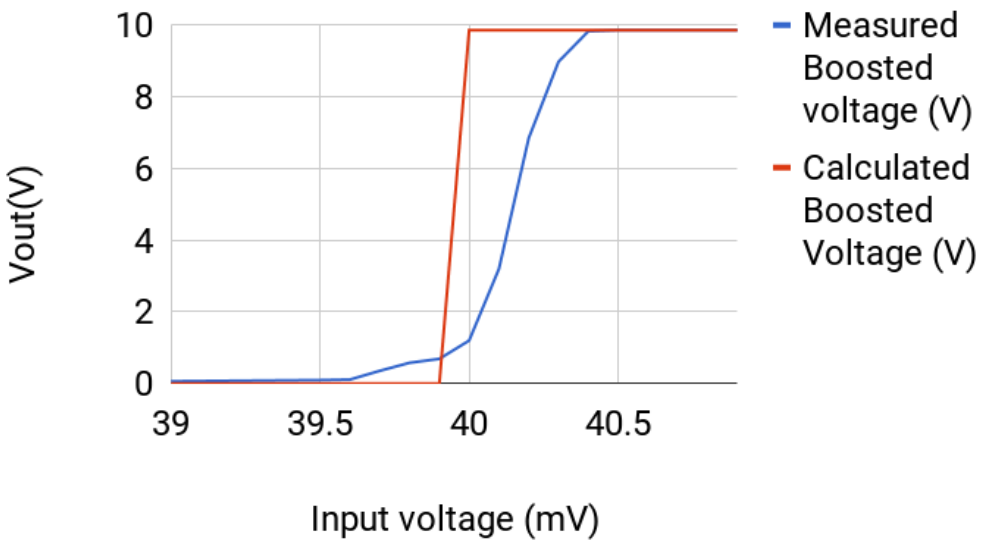

Figure 2.7: Graphical plot of output vs. input voltages of DC-DC booster

2.4 Step-down Converter

The voltage at the output of the DC-DC booster is susceptible to loading, from any subsequent stages of the system. In addition, the voltage output at the booster (9.8 V DC) is too high to operate the End Device. Hence, we seek to step down the DC voltage to obtain a safe level (3.3 V) that can be used to power up the device.

2.4.1 Resistive Divider

The simplest method to step down voltage is using a resistive divider. The output of a simple resistive divider will change according to the electric current supplied to its load. The effective source impedance as seen from a divider of Z_1 and Z_2, as shown in 2.8 [4], will be Z_1 in parallel with Z_2. Load sensitivity can be decreased by reducing the impedance of both parts of the divider, though this increases the divider's quiescent input current and results in higher power consumption (and wasted energy) in the divider.

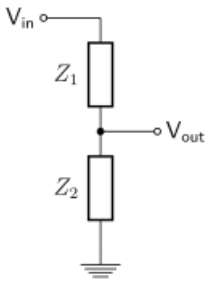

Figure 2.8: Resistive divider (Reprinted from [4])

2.4.2 Low Dropout Regulator

An alternative device often used to step down voltages and step up energy is the Low Drop-out Voltage regulator (LDO). This option was explored, but found sub-optimal, as the dropout voltage (difference between input and output voltage) was very high in the proposed system (6.5 V), which resulted in power inefficiency, leading to very small, unusable current output. A schematic of an LDO is shown in 2.9 [5].

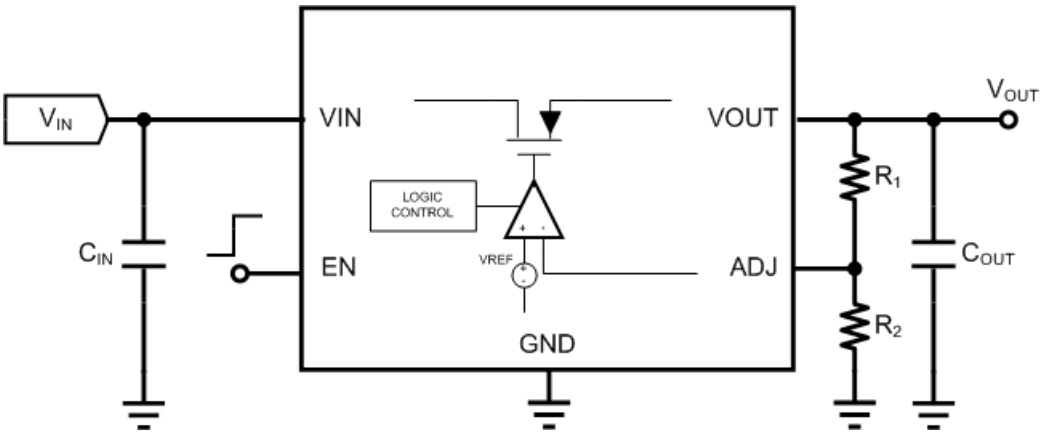

Figure 2.9: Low drop-out regulator (Reprinted from [5])

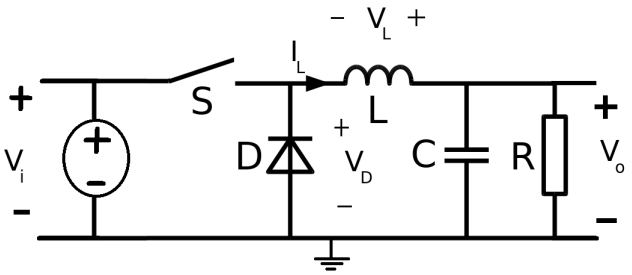

Figure 2.10: Buck converter schematic (Reprinted from [6])

2.4.3 Buck Converter

A step-down (buck) converter is a DC-DC converter which steps down voltage seen at its input. Buck converters have a diode and a MOSFET switch. The switching action of the Buck converter gives rise to a phenomenon called Voltage Ripple, which is undesired. To minimize voltage ripple, capacitive-inductive filters are used at the output side of the converter. A schematic of a Buck converter is shown in 2.10 [6]

Although Buck converters are used for their high efficiency, they also suffer from several sources of power losses and inefficiency

1. Conduction losses: These depend on the load and are caused by the current flowing through the circuit, during regular operation.

    - Power drop across the Resistance (R) when the transistor switch S is conducting (I^2R losses).
    - Diode forward voltage drop (typically 0.7 V)
    - Resistance arising due to imperfect Inductor winding
    - Capacitor's ESR (equivalent series resistance)

2. Switching losses (due to non-ideal switching operations):

- Switching frequency, f_{switch}*CV^2 loss
- Loss due to 'Reverse latency'.
- Losses arising from driving the MOSFET gate (due to finite input impedance).
- Transistor leakage current losses.

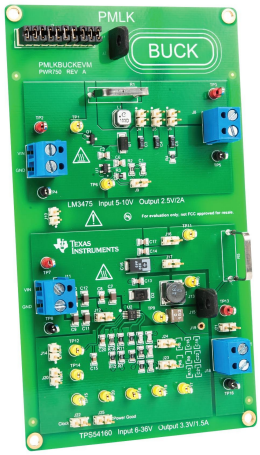

Figure 2.11: TI TPS54160 Buck converter (Reprinted from [7])

In the proposed system, TI's TPS54160 Buck converter, shown in Fig. 2.11 [7]. is used to step down the 9.8V provided the booster to 3.3 V. The device is capable of accepting input voltages from 6 V to 36 V and step them down to 3.3 V.

2.5 End Device

The End Device receives piezoelectric vibrations as analog signals and transmits it after converting the signal to digital [8]. We have used the EZ4300-RF2500, which has the MSP430F2274 microcontroller and CC2500 transceiver on board. The Analog to Digital conversion process happens inside MSP430F2274 Microcontroller. The controller's pins have been programmed to accept an analog input from a source, digitize and process it.

The CC2500 works on the SimpliciTI protocol. The CC2500 transceiver advertizes (scans) for an Access Point (AP) in range. Once an AP is detected, it will transmit all the stored data, wireless-ly,

The End Device can be seen in Figure 2.12 [8]. The End Device code is to be downloaded via a USB port, with the appropriate Minimal RF Interface (MRFI) code. The ED code is in Appendix A and the MRFI code is in Appendix C.

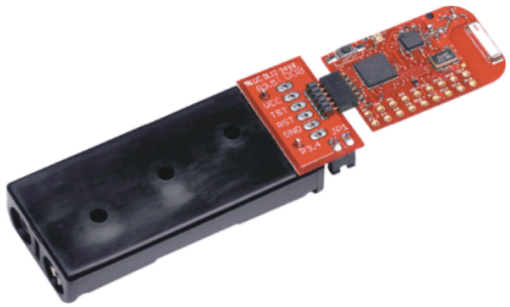

Figure 2.12: End Device (Reprinted from [8])

2.5.1 MSP430F2274 Microcontroller

The MSP430F2274 is the microcontroller embedded on the End Device. The device has a 16-bit RISC CPU and 16-bit registers. It has two built-in 16-bit timers, a universal serial communication interface, a 10-bit A/D converter (ADC10) with a data transfer controller (DTC), two general-purpose operational amplifiers in the MSP430F2274 devices, and 16 I/O pins [9].

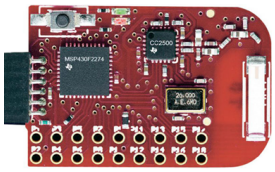

Figure 2.13: Pin diagram of MSP430 controller (Reprinted from ([9])

Refer the picture of the pins on the End Device in Fig 2.13. The MSP430F2274 takes analog input at Pin 3 (with a ground reference at Pin 12), digitizes it and then sends it to the CC2500 for transmission, when an Access Point is in range.

2.5.2 CC2500 Wireless Transceiver

The transceiver used in the proposed system is the TI CC2500, running on the SimpliciTI Protocol - a low-power RF network protocol optimized for small-sized RF networks having a fixed of nodes communicating among each other or through an access point.

The baseband modem of the CC2500 supports several modulation formats and has a configurable data rate, with a maximum of 500 kBaud. In the proposed system, the CC2500 is used together with the MSP430F2274 controller on the ED/AP [19].

There are three modes in which the CC2500 Transceiver operates:

1. Advertising mode:

   This is the default mode of operation of the Transmitter on the End device. While in this mode, the device continuously looks for an Access Point to pair with. As soon as an AP is detected, it is ready to transmit all the data it has been acquiring through its inputs and processing through its on-board ADC in the MSP430F2274.

2. Transmit mode:

   The End device enters this mode once an Access Point is detected and ready for data transmission. The ED will transmit all the digitized data on its memory onto

the AP as long as it is in range. This mode requires 1 mA at 3 V, at a rate of 1.2 kbps. The establishment of a connection between the ED and the AP is signaled by the blinking of one of the LEDs on the AP. Once the connection is lost, the blinking stops and restarts if the connection is restored.

3. Sleep mode:

   This mode has the least power dissipation. In this mode, the ED is not actively looking for an AP to pair with. It simply takes in analog data, processes it, and stores it in memory. This mode typically operates at 390μA at 3 V.

2.5.3 Comparison With Other Controllers And Transceivers

Table 2.3: Comparison with protocols of previous works

	Motorola 68HC11	Hitachi H8-329	AVR-8515	Cygnal 8051	PIC-16F73	Atmel AT-90S8515	Renesas H8-4096F	EZ4300-RF2500
A/D Channels	8	4	1	3	5	1	Multiple	6
A/D Resolution	16-bit	16-bit	16-bit	16-bit	8-bit	16-bit	10-bit	10-bit
Program Memory	16 kB	32 kB	8 kB	2 kB	4 kB	8 kB	128 kB	16 MB
Data Memory	32 kB		32 kB	128 kB	192 kB	512 kB	2 MB	32 kB + 256B flash
Radio	Proxim ProxLink	Radiometrix	Proxim RangeLan2	Ericsson Bluetooth	Realtek RTL-8019AS	Proxim Range LAN2	Phillips Bluetooth	Chipcon CC2500
Outdoor Range	300 m	300 m	300 m	10 m	500 m	300 m	50 m	18 m
Data Rate	19.2 kbps	40 kbps	1.6 Mbps		10 kbps	1.6 Mbps		1.2 kbps

The biggest reason to choose the EZ4300-RF2500 as the End Device is its high Program Memory size. This was a crucial metric, as the codes for the End Device (with the MRFI code), the Access Point, along with the other codes in the environment are heavily memory intensive, and the other End Device architectures would not have supported a code of this size.

2.6 Access Point

The Access Point is a device that is used to collect data from the End Device. The Access Point device can also be used for End Device functionality, i.e., it can itself instantiate sensors or actuators in the network, by virtue of having the MSP430F2274 and the CC2500 embedded on-board. It comes with USB compatibility, in order to be able to display and log the collected data onto a computer (see Fig 2.14). The Access Point code is to be downloaded via a USB port, which is in Appendix B of this document.

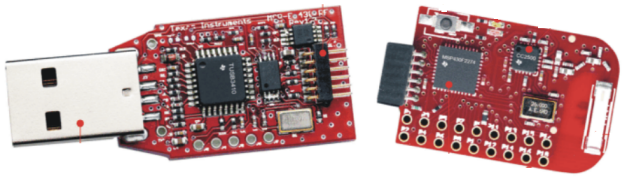

Figure 2.14: Access Point. (Reprinted from [10])

When an Access Point is in range of an End Device, The association between two applications, called linking, takes place. A connection-based object is created through which other End Devices (or Range Extenders) can send messages. Connections are always a bi-directional, indicated by blinking of on-board LEDs. In the proposed system, we've

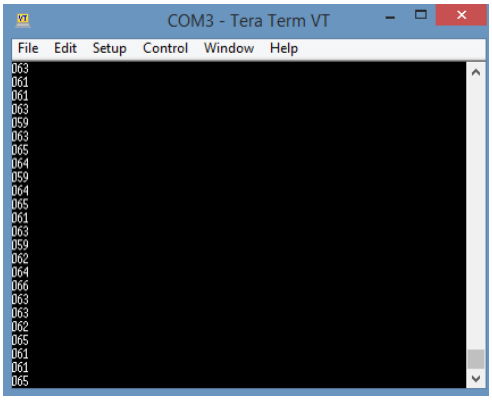

Figure 2.15: Screenshot of data collected at Access Point, on laptop

disabled the blinking of the LEDs on the ED in order to conserve power. Therefore, the indicator of an established connection is the blinking of the LEDs solely on the AP.

The Access Point (AP) is a device usually in a car/control station, which collects and stores digital data that it receives from the ED. This is ultimately, our useful data. The SimpliciTI protocol allows for only one Access Point per system. A screenshot of the data displayed on a computer screen, collected by the Access Point device can be seen in Figure 2.15 [10].

3. SYSTEM INTEGRATION

In previous sections, it was stated that continuous power supply must be provided to the system. The functions of the End Device and the Access Point are also highlighted. This section discusses the integration of the entire system. In the system that will be embedded below the road surface, piezoelectric devices will be used to gauge vibrations from oncoming cars and vehicles. We are using analog inputs to simulate piezoelectric vibrations. These functions are generated using NI ELVIS.

In the lab setup, The temperature gradient across the plates of the TEG is maintained by heating one plate using a heater, and using an ice pack to cool the other plate (Fig. 3.1 and Fig. 3.2).

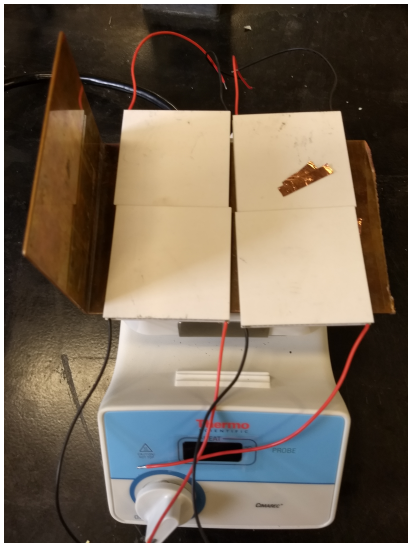

Figure 3.1: Four TEGs mounted on heat source

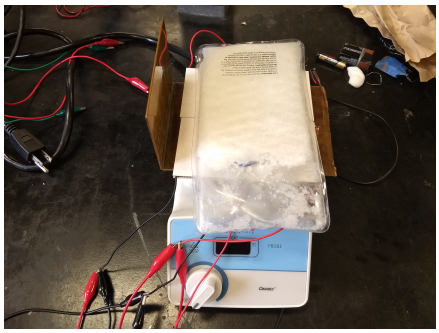

Figure 3.2: TEGs mounted with heat sink

The output from the TEGs is connected to the booster, and subsequently to the Buck converter, finally powering up the End Device (Fig. 3.3).

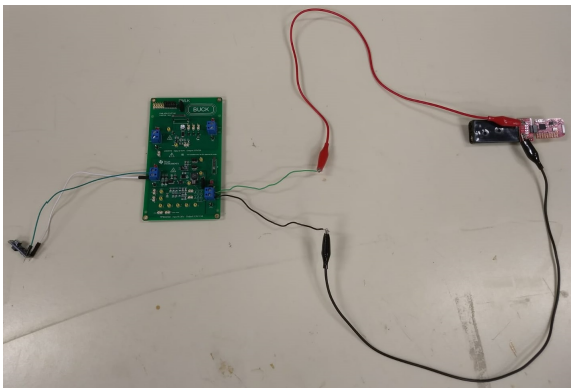

Figure 3.3: DC Booster output going to Buck converter input

3.1 Turn-on Temperature

The Turn-on Temperature T_{th} is the temperature gradient across the plates of the TEG at which the entire loaded system turns on and starts operating. It is a function of the number of TEG devices in parallel, as each new device in parallel provides more current,

allowing the loaded setup to reach steady state at a lower temperature gradient. Therefore, the Turn-on temperature is inversely proportional to the number of TEGs in parallel.

In successive lab trials, TEGs were provided a temperature gradient (by the heater and heat sink), the thermally generated voltage was boosted and regulated and connected to the fully loaded setup.The temperature at which the setup starts operating, was noted. This was performed for 1,2,3 and 4 TEGs in parallel. The results are tabulated in 3.3 and graphically illustrated in Fig 3.4.

Table 3.1: No. of TEG plates in parallel vs. T_{th} (^{0}C)

No. of TEG plates in parallel	T_{th} (^{0}C)
1	12.1
2	5.4
3	3.3
4	2.1

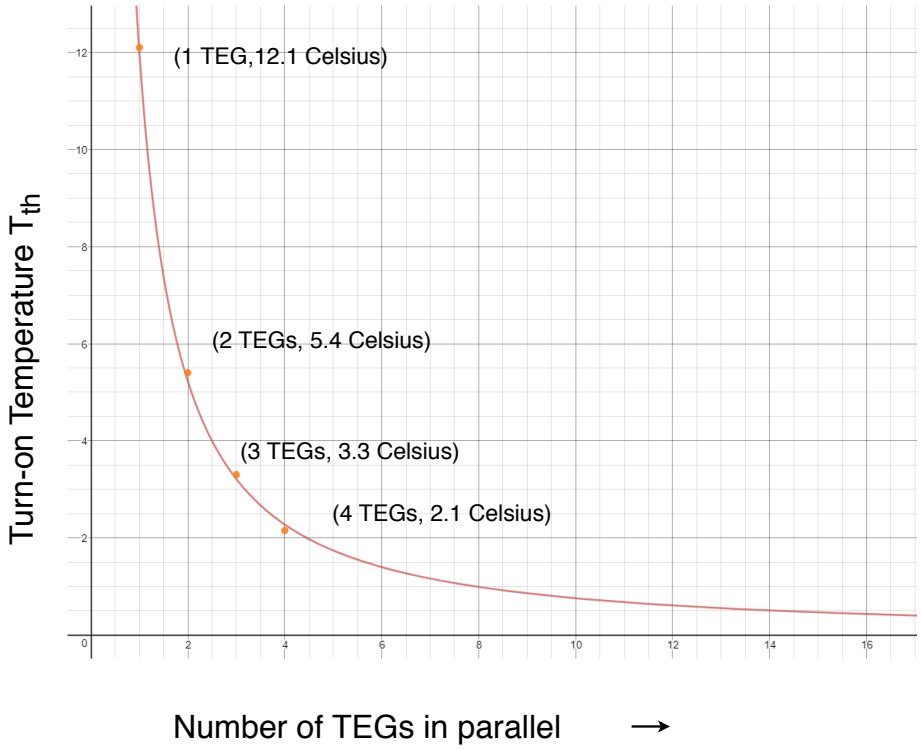

Figure 3.4: No. of TEG plates in parallel vs. Turn-on Temperature (^{0}C)

After plotting the Turn-on temperature vs. number of TEGs in parallel as seen in Fig 3.4 and curve fitting the data points, the relation between number of number of TEGs and Turn-on temperature can be approximated as follows:

$$T_{th} = \frac{12}{n^{1.2}} \tag{3.1}$$

3.1.1 Loaded Buck Converter Input Voltage

The voltage at the input of the loaded buck converter is a function of the temperature gradient ΔT, the Turn-on temperature T_{th} (and therefore, the number of TEGs) and the number of Thermoelectric elements in the generator (N). A graphical depiction is shown

in Fig 3.5.

The voltage at the input of the loaded buck converter as a function of the temperature gradient across its plates, and the turn-on temperature is given by,

$$V_{buck,in} = \begin{cases} 1.004^{\Delta T} * (\frac{\Delta T * N}{211} + 1.77)V & if \quad 0.6 < \Delta T < T_{Th} \\ 9.86V & if \quad \Delta T > T_{Th} \end{cases} \tag{3.2}$$

where ΔT is the temperature gradient, T_{th} is the turn-on temperature and N is the number of Thermoelectric elements in the TEG, 287 in the proposed system.

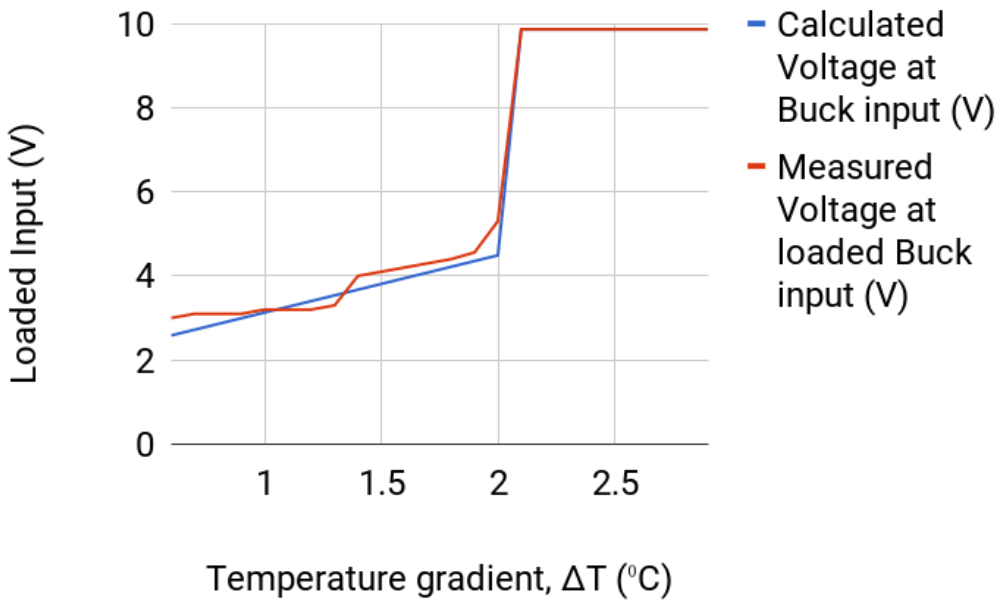

Figure 3.5: Actual vs. Expected voltage at input of loaded Buck converter vs. Temperature, while using 4 TEGs

3.1.2 Loaded Buck Converter Output Voltage

The voltage at the output of the loaded buck converter is a function of the temperature gradient ΔT, the Turn-on temperature T_{th} (and therefore, the number of TEGs) and the number of Thermoelectric elements in the generator (N). A graphical depiction of the input voltage vs. the Temperature gradient, while powered up by four TEGs is shown in Fig 3.6. A comparison of loaded input and output voltages, when the system is powered up by four TEGs is shown in Fig 3.7, confirming that the sharp rise in voltage co-incides with the turn-on temperature. A tabular arrangement of Output Voltage, Current and Power, along with increasing Temperature difference (and therefore input voltage) can be seen in Table 3.2.

The voltage at the input of the loaded buck converter as a function of the temperature gradient across its plates, and the turn-on temperature is given by,

$$V_{buck,out} = \begin{cases} 1.004^{\Delta T} * (\frac{\Delta T * N}{400} + 0.04)V & if \quad 0.6 < \Delta T < T_{Th} \\ 3.318V & if \quad \Delta T > T_{Th} \end{cases} \tag{3.3}$$

where ΔT is the temperature gradient, T_{th} is the turn-on temperature and N is the number of Thermoelectric elements in the TEG, 287 in the proposed system.

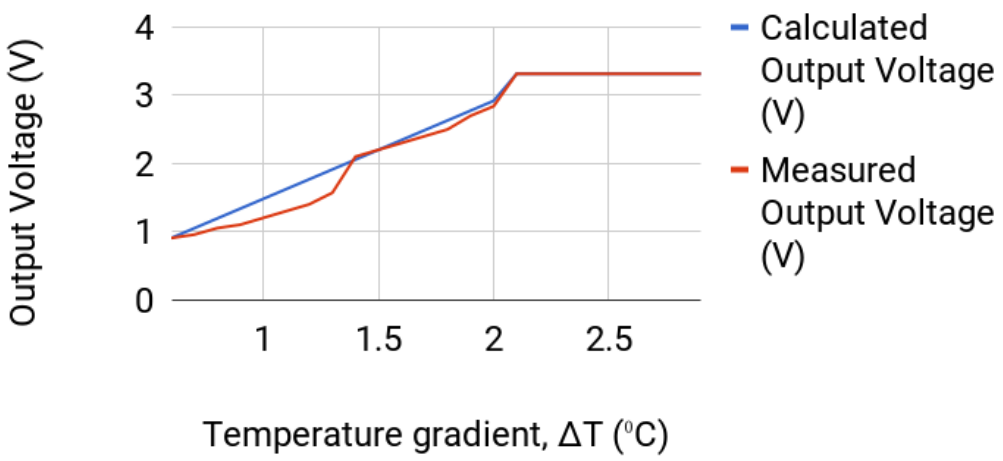

Figure 3.6: Ideal and Real voltage at output of loaded Buck converter vs. Temperature, while using 4 TEGs

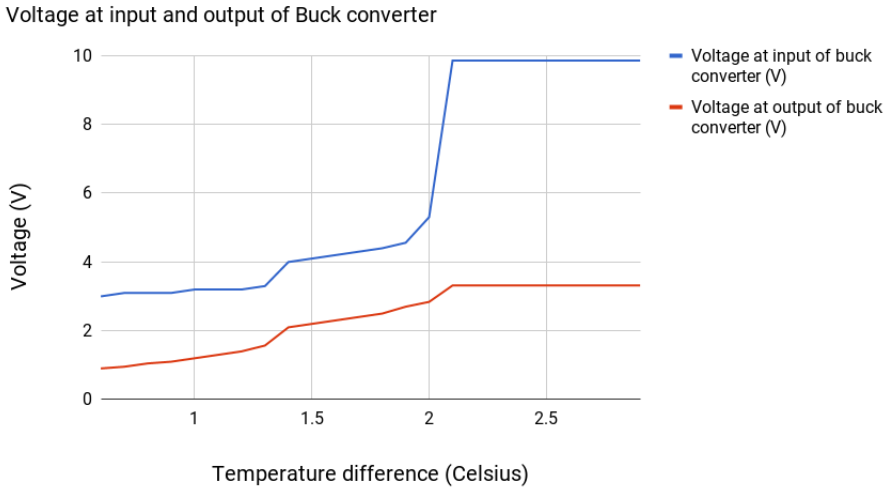

Figure 3.7: Voltage at input and output of loaded buck converter vs Temperature, while using 4 TEGs

Table 3.2: Output voltages, current and power in fully loaded condition

Plate Temperature (Celsius)	Sink Temperature (Celsius)	Temperature difference (Celsius)	Voltage at input of Buck converter (V)	Voltage at output of Buck converter (V)	Output Current (mA)	Output Power (mW)
27.7	27.1	0.6	3.04	0.904	0.274	0.248
27.9	27.2	0.7	3.11	0.954	0.289	0.276
28.1	27.3	0.8	3.16	1.052	0.318	0.334
28.5	27.5	1	3.20	1.24	0.363	0.436
28.7	27.6	1.1	3.21	1.35	0.394	0.512
29.1	27.8	1.3	3.31	1.57	0.477	0.747
29.3	27.9	1.4	4.000	2.12	0.636	1.336
29.5	28.0	1.5	4.113	2.23	0.667	1.467
29.7	28.1	1.6	4.201	2.34	0.697	1.603
29.9	28.2	1.7	4.312	2.43	0.727	1.745
30.3	28.4	1.9	4.560	2.71	0.818	2.209
30.5	28.5	2	5.304	2.848	0.860	2.444
30.7	28.6	2.1	9.86	3.318	1	3.318
30.9	28.7	2.2	9.86	3.318	1	3.318
31.3	28.9	2.4	9.86	3.318	1	3.318
31.7	29.1	2.6	9.86	3.318	1	3.318
32.3	29.4	2.9	9.86	3.318	1	3.318

3.2 Ensuring Continuous Power Supply

During the day, the metal plate closer to the surface of the road will have a higher temperature than one which is deeper down. During the night, the metal plate closer to the surface cools off faster than the one deeper in the ground. Also, during Day/Night transitions, the temperature differences have a zero-crossing, leading to a period of time where the temperature difference is lesser than the Turn-on Temperature, T_{th}, resulting in zero voltage at the output of the TEG. This would turn off the device. To overcome this, a parallel combination of four TEGs is kept at two different depths (see Fig 3.8). Their outputs each go to a DC-DC booster. The voltage from these two boosters is summed up (in series) and is provided as an input to the Buck converter, which powers up the End device (See Fig 3.9). This is done to ensure that, even if the temperature difference between one depth and the surface has a value lesser than T_{th}, it is unlikely that the other depth will have a value lesser than T_{th} at the same time (Fig 3.8). However, during the course of a year over which the temperature data was recorded, there are several time durations when the temperature difference between both depths (relative to the surface) goes below T_{th}. In this case, the device will be powered off and unable to accept input data. However, within those time periods, under the assumption that the temperature changes linearly, the time periods where temperatures at least one of the two devices is above T_{th} is calculated. This time is subtracted from the sum total of all time intervals wherein the temperature of both plates goes below the turn-on temperature, to obtain that sub-interval wherein data cannot be collected/transmitted. This is significantly lesser than if the entire hour was considered.

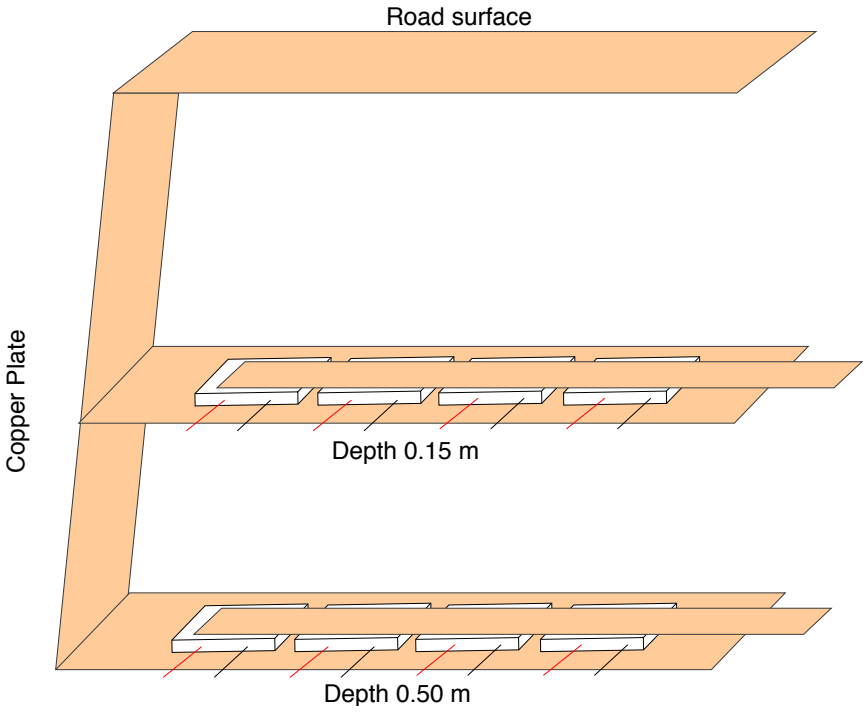

Figure 3.8: Illustration of TEGs at two different depths

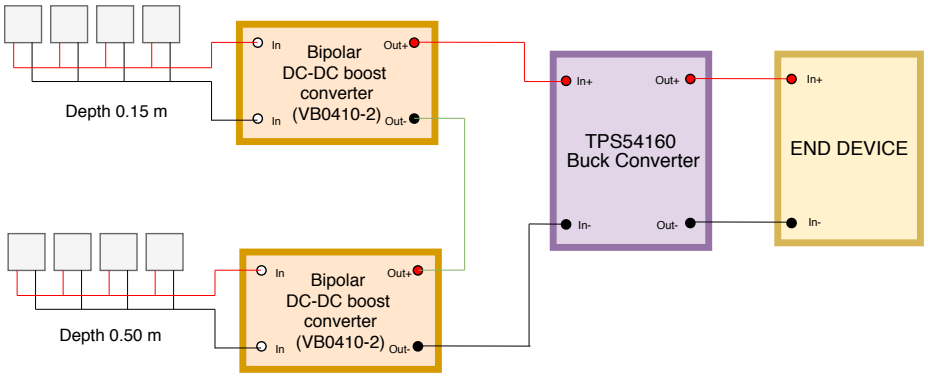

Figure 3.9: Continuous power supply from TEGs at two depths

Using the year-long temperature data (From 1st December, 2014 to 30th November, 2015) collected hourly from Texas and San Antonio, the following calculations were made to compute the Effective off-time at both locations.

1. $Sign_{0.15,0.50}$: Compares the sign of one temperature reading to that of the temperature reading from the previous hour. If they're both of the same sign, 'Sign' returns 0, else returns 1.

$$Sign_{0.15,0.50} = IF(\Delta T_{T1} * \Delta T_{T2} < 0, 1, 0) \tag{3.4}$$

   Here, ΔT_{T2} has the difference in temperature between the plates on surface and that at 0.5m at time T_2 and ΔT_{T1} has the temperature difference at the same depth at T_1.

2. $Sign_{both}$: Compares the change in sign of one temperature reading to that of the temperature reading at another depth. If they're both equal to one, $Sign_{both}$ returns 1, else returns 0. This indicates whether both plates undergo a zero-crossing during a certain time-interval.

$$Sign_{both} = IF(Sign_{0.15} * Sign_{0.50} = 1, 1, 0) \tag{3.5}$$

3. $Zero_{Cross,0.15,0.5}$: Returns the time at which the temperature of the plate goes below T_{th} at 0.15m (or 0.5m), by linear extrapolation, assuming a linear relation between time and temperature for that one hour duration. t_1 and t_2 are the start and end times, $\Delta T_{0.15,T1}$ and $\Delta T_{0.15,t2}$ store the temperature difference at 0.15m at t_1 and t_2

respectively. The exact analogous variables exist for the plates at 0.5m.

$$Zero_{Cross} = if(Sign_{both} = 1, t1+(2.1-\Delta T_{0.15,t1})*(t2-t1)/(\Delta T_{0.15,t2}-\Delta T_{0.15,t1})) \tag{3.6}$$

4. $Time_{zero}$: Returns the difference in minutes between the time of the temperature of one plate going below T_{th} and another.

$$Time_{zero} = abs(Zero_{Cross,0.15} - Zero_{Cross,0.5}) * 60 \tag{3.7}$$

5. $Conditional_{turn-on}$: Returns the time at which one plate crosses the turn-on temperature (ΔT_{th}) before or after its own time of going below the turn-on temperature. This formula is applied to both plates.

$$\begin{aligned} Conditional_{turn-on} = if(AND(Sign_{both} = 1, OR(\Delta T_{0.50,t1} > \Delta T_{th}, \Delta T_{0.50,t2} \\ > \Delta T_{th})), t1 + (\Delta T_{th} - \Delta T_{0.50,t1}) * (t2 - t1)/ \\ (\Delta T_{0.50,t2} - \Delta T_{0.50,t1}), \text{""}) \end{aligned} \tag{3.8}$$

6. $Time_{turnon1,2}$: Time in minutes that one plate is on, while the other is off. This returns zero if one plate is off throughout the off time of the other plate, and returns the on-time in minutes, otherwise.

$$\begin{aligned} Time_{turnon1,2} = & if(Conditional_{turn-on,1} < Time_{zero,2}, \\ & 60 * (Conditional_{turn-on,1} - Time_{zero,2}), 0) \end{aligned} \tag{3.9}$$

7. Eff_{off}: Total time in the considered one hour time interval where both devices are off. This is computed by subtracting Time difference between turn-off times of the two plates and the conditional turn-on times of both plates. If the latter is larger than the former, it returns 0, because it implies that the device was on for longer than it was off. Another way of looking at it is that, while one plate was approaching a zero-crossing, the other plate was on, and vice versa. Eff_{off} can range from 0 to 60 minutes.

$$Eff_{off} = max(Time_{zero} - Time_{turnon,1} - Time_{turnon,2}, 0) \tag{3.10}$$

In figure 3.10, both plates have a zero-crossing within the same interval. Also, while the temperature on one plate is below the turn-on temperature, the temperature on the other plate never exceeds the turn-on temperature. In this case, there is no conditional turn-on for either plates.

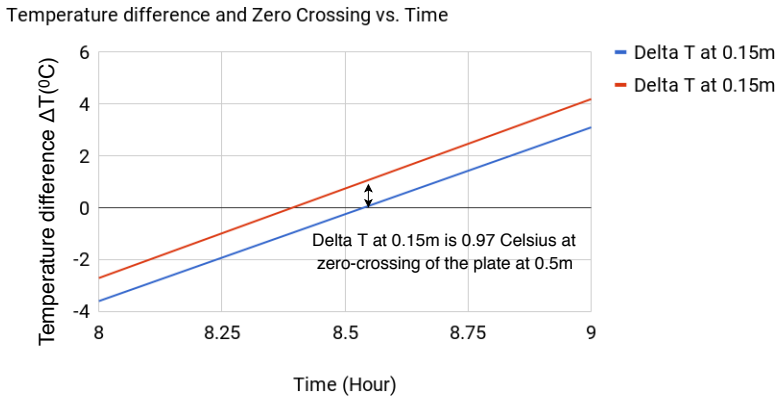

Figure 3.10: Zero crossing with both plates off

In figure 3.11, both plates also have a zero-crossing within the same interval. However, while the temperature on one plate is below the turn-on temperature, the temperature on the other plate exceeds the turn-on temperature, for a few minutes. This time interval is subtracted from the time duration (one hour) in which the temperature gradient at both plates goes below the turn-on temperature.

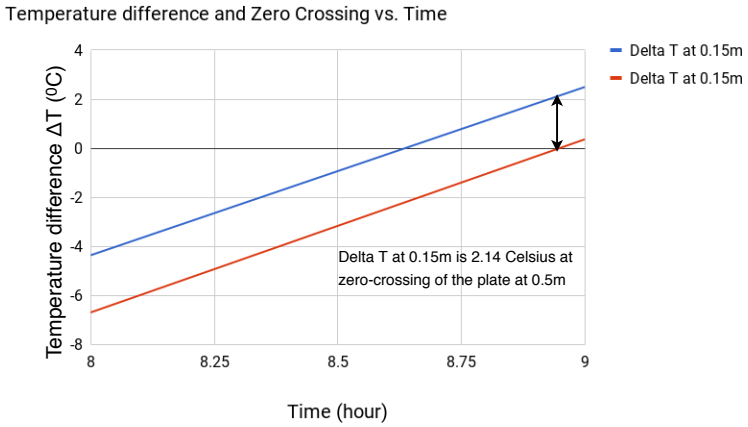

Figure 3.11: Zero crossing with at least one plate on

For a time period of one year, the effective off-times are summed. Table 3.3 shows the total number of hours the device would be switched off, for the duration of one year between December 2014 and 30th November 2015, for San Antonio and Minnesota. It is evident that the switch-off time falls with increasing number of TEG plates. A graphical depiction is seen in Fig 3.12

Table 3.3: No. of TEG plates in parallel vs. Total time of non-operation (hours)

No. of TEG plates in parallel	Total hours of non-operation in San Antonio (ΣEff_{off})	Total hours of non-operation in Minnesota (ΣEff_{off})
1	2815	2410
2	508	368
3	115	98
4	30	18

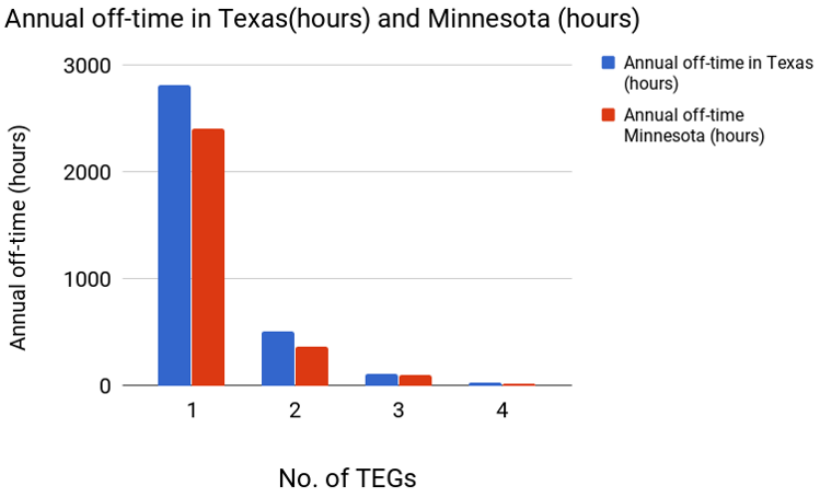

Figure 3.12: Annual off-time in Minnesota and San Antonio

3.3 Operating Rang, Timing and Power Considerations

The CC2500 transceiver (on the EZ4300-RF2500 chip) used in the proposed work transmits data at a minimum rate of 1.2 kbps. The MSP430F2274 microcontroller has a 10-bit ADC which stores the piezoelectric impulse from an oncoming car into 3 bytes of data. Therefore, each passing car (two-axle) provides 6 bytes or 48 bits of data.

Assuming that the Access Point is in a car moving at the speed limit of the road, the total data it can accept from the End device can be computed as follows:

$$Speed_{Max} : 100mph = 44.7ms^{-1}. \tag{3.11}$$

This is the worst case scenario, as it is the condition for minimum time for data transfer. A higher data rate is required for a greater range, but this comes at the cost of a higher power consumption. Range (R) scales with Power (P) as seen in Fig 3.13. It is an almost linear relation.

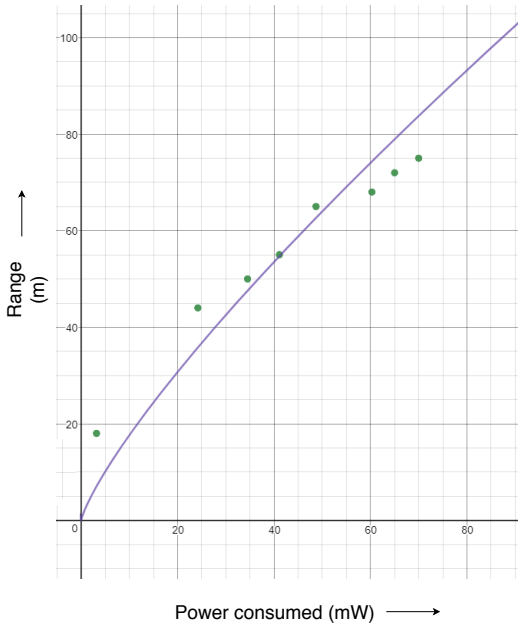

Figure 3.13: Range vs. Power

After plotting the Range vs. Power as seen in Fig 3.13 and curve fitting the data points, the relation between Range of the device and Power consumption can be approximated as follows:

$$R = 3.3 * P^{0.8} \tag{3.12}$$

It is instructive to have a direct mapping between the power consumption and the amount of data that can be transferred to the Access Point device, in order to gain insight on how frequently the Access Point should pass over the End Device. This is seen in Table 3.4, and the same is plotted in Fig.3.14.

Table 3.4: Power consumption vs. Range

Power Consumption (mW)	Range (m)	Maximum transmission time (s)	Data that can be transferred (kb)	No. of cars whose data can be transferred
3.18	18	0.81	0.966	20
24.2	44	1.97	98.43	2050
34.5	50	2.24	223.7	4660
41.1	55	2.46	369.1	7690
48.7	65	2.91	727.1	15147
60.3	68	3.04	912.8	19015
65.	72	3.22	1288.6	26845
70.0	75	3.36	1677.8	34955

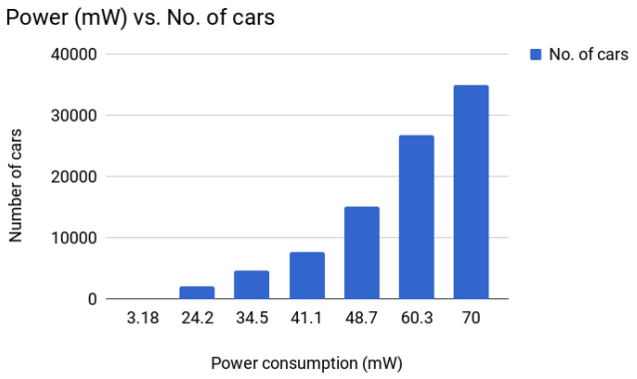

Figure 3.14: Power vs. number of cars

3.4 Comparison With Previous Works

Several other works were compared with each other, in terms of output power, data bit rate, the nature of the power source, their ability for continuous operation and their use of batteries. A comparative table of this work against four others, is shown in Table 3.5.

Table 3.5: Comparison with previous works

Work	Power source	Power output	Data Rate	Use of batteries	Continuous Operation
Dondi et. al [13]	Piezoelectric	0.845 mW	22 bps	yes	yes
Zhou et. al [12]	Piezoelectric	2.9 mW	50 kbps	yes	no
Hassan et. al [14]	Solar	120 mW	1 kbps	yes	yes
Yuan and Fang [15]	Magneto-restrictive	0.576 mW		no	yes
This work	Thermo-electric	3.3 mW	1.2 kbps	no	yes

Table 3.5 is intended to illustrate various sources of power that has been employed for Structural Health Monitoring. It also lays down a comparison of the performance characteristics of the afore-mentioned methods. It can be seen that most works ([13], [12], and [14] use batteries as a power backup during off-times (typically during night, for solar-powered SHM solutions). Some works also do not provide continuous operation ([12]). The proposed system overcomes these limitations by providing continuous, uninterrupted

power supply, while also providing a data rate of 1.2 kbps over a range of 18m.

4. SUMMARY, CONCLUSIONS AND FUTURE WORK

The aim of the proposed work was the development of a self-powered sensor, storage and transmission system intended for the use of Structural Health Monitoring. To that end, the following was demonstrated: The End device was successfully powered up using thermally generated electricity. We are able to read analog data at the End device, look for an access point and transmit data to it wirelessly. The received data is readable on a laptop attached to the Access Point. Thus, an end-to-end, batteryless and self-powered system to gauge vibrations from oncoming vehicles and to store and transmit it wirelessly, is built.

In order to run the system at maximum data rate of 500 kbps, the current required is 22-25 times more than is currently available. Therefore, 90-100 TEGs would be required in parallel. One TEG is 62 mm x 62 mm x 5.3 mm in dimension. 100 of these, along with the Booster would fit in a box of 7 inch x 7 inch x 7 inch. Since two such TEG setups are required at two different depths, two such boxes will be needed. These, along with the Buck converter and the End Device can be easily embedded under the surface of the road.

How the functionality of the device is affected after placing it underneath the road, remains to be seen. The Range vs. Power trade-off is a useful problem to solve. The implementation of Range Extenders in order to overcome this shortcoming is worthy of consideration. The system power also limited the performance and the range of the device. Collecting data using a drive-by car is unfeasible, given the small time-frame of data transfer at high speeds. Lower power solutions based on ZigBee/Bluetooth Low Energy (BLE) protocols can be explored. BLE is also better than ZigBee as the latter is optimized for star/mesh topology, consuming greater current. BLE supports P2P topology, and consumed lesser current.

REFERENCES

[1] "T.X.L-287-03.Z T.E.G. 62 X 62mm. Custom Thermoelectric. N.p., n.d. Web. 11 June 2018.." https://customthermoelectric.com/txl-287-03z-teg-62-x-62mm.html.

[2] CyrilB commonswiki assumed (based on copyright claims)., "Dc booster." https://commons.wikimedia.org/w/index.php?curid=629013.

[3] "ELC-VB0410-2 Bipolar Voltage Booster 40mV. Custom Thermoelectric. N.p., n.d. Web. 11 June 2018.." https://customthermoelectric.com/elc-vb0410-2-bipolar-voltage-booster-40mv.html.

[4] Krishnavedala - Own work, CC0, "Resistive divider." https://commons.wikimedia.org/w/index.php?curid=36347358.

[5] en:Lagrangian - Own work, Public Domain, "LDO." https://commons.wikimedia.org/w/index.php?curid=22345641.

[6] P. D. TyIzael Self; conversion of File:Buck_circuit.gif by Zack112358., "Dc buck converter." https://commons.wikimedia.org/w/index.php?curid=8687723.

[7] "Texas Instruments PMLKBUCKEVMB Experiment Board, Buck, TPS54160, LM3475, TI-PMLK Buck Experiment Lab Book." https://www.tanotis.com/products/texas-instruments-pmlkbuckevmb\
\-experiment-board-buck-tps54160-lm3475-ti-pmlk-buck-experiment-lab-book?variant=40460420368.

[8] "A brief Tutorial On SimpliciTI 1.1.1." http://www.ti.com/tool/ez430-rf2500t.

[9] Texas Instruments, "MSP430F2274." http://www.ti.com/product/msp430f2274.

[10] "Access Point." http://processors.wiki.ti.com/index.php/EZ430-RF2500.

[11] "US Department of transportation and Federal Highway Administration. Number of motor vehicles registered in the United States from 1990 to 2016 (in 1,000s)." https://www.statista.com/statistics/183505/number-of-vehicles-in-the-united-states-since-1990/.

[12] Zhou, D., Na, K., Ha, D.S., and Inman, Daniel, "A self-powered wireless sensor node for structural health monitoring," in *SPIE Smart Structures and Materials + Nondestructive Evaluation and Health Monitoring*, pp. 2–13, SPIE, 2010.

[13] Dondi, D., Napoletano, G., Bertacchini, A., Larcher L., and Pavan, P., "A WSN system powered by vibrations to improve safety of machinery with trailer," pp. 28–31, IEEE, 2012.

[14] Hassan, M., Man, S., Ng, C., Bermak, A., Chang, C., "Development of energy harvested wireless sensing node for structural health monitoring," pp. 22–27, IEEE, 2012.

[15] F. Yuan and F. Zhang, "Vibration energy harvesting by magnetostrictive material," vol. 17, pp. 2–15, 2008.

[16] Authors of Encyclopedia Britannica, "Seeback Effect." https://www.britannica.com/science/Seebeck-effect.

[17] Unknown author, "Seeback Effect." https://searchnetworking.techtarget.com/definition/Seebeck-effect.

[18] Saida, M., Zaibi, G., Samet, M., and Kachouri, A., "A new design of thermoelectric generator for health monitoring," *International Conference on Smart, Monitored and Controlled Cities (SM2C),*, 2017.

[19] Texas Instruments, "CC2500." http://www.ti.com/product/CC2500.

APPENDIX A

END DEVICE CODE

```
#include "bsp.h"
#include "mrfi.h"
#include "nwk_types.h"
#include "nwk_api.h"
#include "bsp_leds.h"
#include "bsp_buttons.h"
#include "vlo_rand.h"
//#include  "msp430g2253.h"

/*------------------------------------------------------------------------------
 *------------------------------------------------------------------------------
/* How many times to try a TX and miss an acknowledge
 before doing a scan */
#define MISSES_IN_A_ROW  2

/*------------------------------------------------------------------------------
 *------------------------------------------------------------------------------
static void linkTo(void);
void createRandomAddress(void);
```

```
__interrupt void ADC10_ISR(void);
__interrupt void Timer_A (void);

/*-------------------------------------------------------
* Globals
-------------------------------------------------------*/
static linkID_t sLinkID1 = 0;
/* Temperature offset set at production */
volatile int * tempOffset = (int *)0x10F4;
/* Initialize radio address location */
char * Flash_Addr = (char *)0x10F0;
/* Work loop semaphores */
static volatile uint8_t sSelfMeasureSem = 0;


// Global variables
int adc[10] = {0}; //Sets up an array of 10 integers
and zero's the values
int avg_adc = 0;
//MRFI_Init();

// Function prototypes
void adc_Setup();
void adc_Sam10();
```

```
/*-------------------------------------------------------------
 * Main
 *-------------------------------------------------------------
void main (void)
{
  addr_t lAddr;

  /* Initialize board-specific hardware */
  BSP_Init();
  //WDTCTL = WDTPW + WDTHOLD;          // Stop WDT
  adc_Setup();

  /* Check flash for previously stored address */
  if(Flash_Addr[0] == 0xFF && Flash_Addr[1] == 0xFF &&
     Flash_Addr[2] == 0xFF && Flash_Addr[3] == 0xFF )
  {
    createRandomAddress(); // Create and store a new
    random address
  }

  /* Read out address from flash */
  lAddr.addr[0] = Flash_Addr[0];
  lAddr.addr[1] = Flash_Addr[1];
  lAddr.addr[2] = Flash_Addr[2];
  lAddr.addr[3] = Flash_Addr[3];
```

```
/* Tell network stack the device address */
SMPL_Ioctl(IOCTL_OBJ_ADDR, IOCTL_ACT_SET, &lAddr);

/* Initialize TimerA and oscillator */
BCSCTL3 |= LFXT1S_2;      // LFXT1 = VLO
TACCTL0 = CCIE;             // TACCR0 interrupt enabled
TACCR0 = 24000;           // ~ 1 sec
TACTL = TASSEL_1 + MC_1;          // ACLK, upmode


while (SMPL_SUCCESS != SMPL_Init(0))
{
  //BSP_TOGGLE_LED1();
  //BSP_TOGGLE_LED2();
  /* Go to sleep (LPM3 with interrupts enabled)
   * Timer A0 interrupt will wake CPU up every second
   to retry initializing
   */
  __bis_SR_register(LPM3_bits+GIE);
  // LPM3 with interrupts enabled
}

/* LEDs on solid to indicate successful join. */
//BSP_TURN_ON_LED1();
```

```
  //BSP_TURN_ON_LED2();

  /* Unconditional link to AP which is listening due
  to successful join. */
  linkTo();

  while(1);
}

static void linkTo()
{
  uint8_t msg[3];
#ifdef APP_AUTO_ACK
  uint8_t misses, done;
#endif

  /* Keep trying to link... */
  while (SMPL_SUCCESS != SMPL_Link(&sLinkID1))
  {
    //BSP_TOGGLE_LED1();
    //BSP_TOGGLE_LED2();
    /* Go to sleep (LPM3 with interrupts enabled)
     * Timer A0 interrupt will wake CPU up every second
     to retry linking
     */
```

```
  __bis_SR_register(LPM3_bits+GIE);
}

/* Turn off LEDs. */
//BSP_TURN_OFF_LED1();
//BSP_TURN_OFF_LED2();



while (1)
{
  /* Go to sleep, waiting for interrupt every second
  to acquire data */
  __bis_SR_register(LPM3_bits);


    adc_Sam10();        // Function call for adc_samp
    // Add all the sampled data and divide by 10 to
    //find average
    avg_adc = ((adc[0]+adc[1]+adc[2]+adc[3]+adc[4]+
    adc[5]+adc[6]+adc[7]+adc[8]+adc[9]) / 10);
    //avg_adc = adc[0];

    msg[2]=(avg_adc/100);
    msg[1]=(avg_adc%100/10);
```

```
        msg[0]=(avg_adc%10);

        //msg[2] = 2;
        //msg[1] = 1;
        //msg[0] = 0;
        /* Get radio ready...awakens in idle state */


#ifdef APP_AUTO_ACK
        /* Request that the AP sends back to confirm
        data transmission
         * Note: Enabling this section more than DOUBLES
         current consumption due to the amount of time
         waiting for the AP to respond
         */
        done = 0;
        while (!done)
        {
          noAck = 0;

          /* Try sending message MISSES_IN_A_ROW
          times looking for ack */
          misses=0;
          for (; misses < MISSES_IN_A_ROW; ++misses)
          {
```

```
    if (SMPL_SUCCESS==(rc=SMPL_SendOpt(sLinkID1,
    msg, sizeof(msg), SMPL_TXOPTION_ACKREQ)))
    {
      /* Message acked. We're done. Toggle LED
      to indicate ack received. */
      BSP_TURN_ON_LED1();
      __delay_cycles(2000);
      BSP_TURN_OFF_LED1();
      break;
    }
    if (SMPL_NO_ACK == rc)
    {
      /* Count ack failures. Could also fail
      becuase of CCA and
       * we don't want to scan in this case.
       */
      noAck++;
    }
  }
  if (MISSES_IN_A_ROW == noAck)
  {
    /* Message not acked */
    BSP_TURN_ON_LED2();
    __delay_cycles(2000);
    BSP_TURN_OFF_LED2();
```

```
#ifdef FREQUENCY_AGILITY
            /* Assume we're on the wrong channel so
            look for channel by
             * using the Ping to initiate a scan when
             it gets no reply. With
             * a successful ping try sending the
             message again. Otherwise,
             * for any error we get we will wait
             until the next button
             * press to try again.
             */
            if (SMPL_SUCCESS != SMPL_Ping(sLinkID1))
            {
              done = 1;
            }
#else
            done = 1;
#endif  /* FREQUENCY_AGILITY */
          }
          else
          {
            /* Got the ack or we don't care. */
            done = 1;
          }
        }
```

```
#else

      /* No AP acknowledgement, just send a single
      message to the AP */
SMPL_SendOpt(sLinkID1,msg,sizeof(msg),TXOPTION_NONE);

#endif /* APP_AUTO_ACK */

      /* Put radio back to sleep */
      SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_RADIO_SLEEP, 0);

      /* Done with measurement, disable measure flag */
      sSelfMeasureSem = 0;
    }
  }


void createRandomAddress()
{
  unsigned int rand, rand2;
  do
  {
    rand = TI_getRandomIntegerFromVLO();
    // first byte can not be 0x00 of 0xFF
  }
```

```
  while( (rand & 0xFF00)==0xFF00 ||
  (rand & 0xFF00)==0x0000 );
  rand2 = TI_getRandomIntegerFromVLO();

  BCSCTL1 = CALBC1_1MHZ;
  // Set DCO to 1MHz
  DCOCTL = CALDCO_1MHZ;
  FCTL2 = FWKEY + FSSEL0 + FN1;
  // MCLK/3 for Flash Timing Generator
  FCTL3 = FWKEY + LOCKA;
  // Clear LOCK & LOCKA bits
  FCTL1 = FWKEY + WRT;
  // Set WRT bit for write operation

  Flash_Addr[0]=(rand>>8) & 0xFF;
  Flash_Addr[1]=rand & 0xFF;
  Flash_Addr[2]=(rand2>>8) & 0xFF;
  Flash_Addr[3]=rand2 & 0xFF;

  FCTL1 = FWKEY;
  // Clear WRT bit
  FCTL3 = FWKEY + LOCKA + LOCK;
  // Set LOCK & LOCKA bit
}
```

```
/*-------------------------------------------------------------
 * ADC10 interrupt service routine
 *-------------------------------------------------------------
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{
  __bic_SR_register_on_exit(CPUOFF);
  // Clear CPUOFF bit from 0(SR)
}

/*-------------------------------------------------------------
 * Timer A0 interrupt service routine
 *-------------------------------------------------------------
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A (void)
{
  sSelfMeasureSem = 1;
  __bic_SR_register_on_exit(LPM3_bits);
  // Clear LPM3 bit from 0(SR)
}

// ADC10 interrupt service routine

// ADC set-up function
void adc_Setup()
```

```
{
    ADC10CTL1 = CONSEQ_2 + INCH_0;
    // Repeat single channel, A0
    ADC10CTL0 = ADC10SHT_2 + MSC + ADC10ON + ADC10IE;
    // Sample & Hold Time + ADC10 ON + Interrupt Enable
    ADC10DTC1 = 0x01;
    // 10 conversions
    ADC10AE0 |= 0x0A;
    // P1.0 ADC option select
}

// ADC sample conversion function
void adc_Sam10()
{
    ADC10CTL0 &= ~ENC;
    // Disable Conversion
    while (ADC10CTL1 & BUSY);
    // Wait if ADC10 busy
    ADC10SA = (int)adc;

    ADC10CTL0 |= ENC + ADC10SC;
    // Enable Conversion and conversion start
    __bis_SR_register(CPUOFF + GIE);
    // Low Power Mode 0, ADC10_ISR
}
```

APPENDIX B

ACCESS POINT CODE

```
#include <string.h>
#include <stdio.h>
#include "bsp.h"
#include "mrfi.h"
#include "bsp_leds.h"
#include "bsp_buttons.h"
#include "nwk_types.h"
#include "nwk_api.h"
#include "nwk_frame.h"
#include "nwk.h"
#include "virtual_com_cmds.h"


/* Frequency Agility helper functions */
static void    checkChangeChannel(void);
static void    changeChannel(void);

__interrupt void ADC10_ISR(void);
__interrupt void Timer_A (void);
```

```
/*---------------------------------------------------------
 * Globals
 *---------------------------------------------------------
/* reserve space for the max possible peer Link IDs */
static linkID_t sLID[NUM_CONNECTIONS] = {0};
static uint8_t  sNumCurrentPeers = 0;

/* callback handler */
static uint8_t sCB(linkID_t);

/* received message handler */
static void processMessage(linkID_t,uint8_t *,uint8_t);

/* work loop semaphores */
static volatile uint8_t sPeerFrameSem = 0;
static volatile uint8_t sJoinSem = 0;
static volatile uint8_t sSelfMeasureSem = 0;

/* blink LEDs when channel changes... */
static volatile uint8_t sBlinky = 0;

/* data for terminal output */
const char splash[] = {"\r\n----------
-----------------------------------------

\r\n      ****\r\n
```

```
****              eZ430-RF2500\r\n      ******o****
Temperature Sensor Network\r\n********_///_****
Copyright 2009\r\n ******/_//_/*****
Texas Instruments
Incorporated\r\n  ** ***(__/*****
All rights reserved.\r\n
*********      SimpliciTI1.1.1\r\n
*****\r\n         ***\r\n-------------------------
-------------------------------\r\n"};
volatile int * tempOffset = (int *)0x10F4;

/*-------------------------------------------------------
 * Frequency Agility support (interference detection)
 *-------------------------------------------------------*/
#ifdef FREQUENCY_AGILITY

#define INTERFERNCE_THRESHOLD_DBM (-70)
#define SSIZE     25
#define IN_A_ROW  3
static int8_t  sSample[SSIZE];
static uint8_t sChannel = 0;

#endif  /* FREQUENCY_AGILITY */

char printing[] = {"\r\nXXX"};
```

```
int value=0;

/*-------------------------------------------------------------
 * Main
 *-------------------------------------------------------------*/
void main (void)
{
  bspIState_t intState;

#ifdef FREQUENCY_AGILITY
  memset(sSample, 0x0, sizeof(sSample));
#endif

  /* Initialize board */
  BSP_Init();

  /* Initialize TimerA and oscillator */
  BCSCTL3 |= LFXT1S_2;
  // LFXT1 = VLO
  TACCTL0 = CCIE;
  // TACCR0 interrupt enabled
  TACCR0 = 12000;
  // ~1 second
  TACTL = TASSEL_1 + MC_1;
  // ACLK, upmode
```

```
/* Initialize serial port */
COM_Init();

//sprintf(printing,"Started");

//Transmit splash screen, network init notification
TXString( (char*)splash, sizeof splash);
TXString( "\r\nInitializing Network....", 26 );

SMPL_Init(sCB);

// network initialized
TXString( "Done\r\n", 6);

/* green and red LEDs on solid to indicate waiting
for a Join. */
BSP_TURN_ON_LED1();
BSP_TURN_ON_LED2();

/* main work loop */
while (1)
{
  /* Wait for the Join semaphore to be set by the
  receipt of a Join frame from
```

```
 * a device that supports an End Device.
 *
 * An external method could be used. A button
 press could be connected
 * to an ISR and the ISR could set a semaphore that
 is checked by a function
 * call here, or a command shell running in support
 of a serial connection
 * could set a semaphore checked by a function
 call.
 */
if (sJoinSem && (sNumCurrentPeers < NUM_CONNECTIONS))
{
  /* listen for a new connection */
  while (1)
  {
    if (SMPL_SUCCESS == SMPL_LinkListen
    (&sLID[sNumCurrentPeers]))
    {
      break;
    }
    /* Implement fail-to-link policy here.
    otherwise, listen again. */
  }
```

```
    sNumCurrentPeers++;

    BSP_ENTER_CRITICAL_SECTION(intState);
    sJoinSem--;
    BSP_EXIT_CRITICAL_SECTION(intState);
}



    /* Get temperature */
    //ADC10CTL1 = INCH_10 + ADC10DIV_4;
    // Temp Sensor ADC10CLK/5
    //ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON
    + ADC10ON + ADC10IE + ADC10SR;
    /* Allow ref voltage to settle for at least
    30us (30us * 8MHz = 240 cycles)
     * See SLAS504D for settling time spec
     */
    //__delay_cycles(240);
    //ADC10CTL0 |= ENC + ADC10SC;
    // Sampling and conversion start
    //__bis_SR_register(CPUOFF + GIE);
    // LPM0 with interrupts enabled
    //results[0] = ADC10MEM;
    // Retrieve result
```

```
//ADC10CTL0 &= ~ENC;

/* Get voltage */
//ADC10CTL1 = INCH_11;
// AVcc/2
//ADC10CTL0 = SREF_1 + ADC10SHT_2 +
REFON + ADC10ON + ADC10IE + REF2_5V;
//__delay_cycles(240);
//ADC10CTL0 |= ENC + ADC10SC;
// Sampling and conversion start
//__bis_SR_register(CPUOFF + GIE);
// LPM0 with interrupts enabled
//results[1] = ADC10MEM;
// Retrieve result

/* Stop and turn off ADC */
//ADC10CTL0 &= ~ENC;
//ADC10CTL0 &= ~(REFON + ADC10ON);

/* oC = ((A10/1024)*1500mV)-986mV)*1/3.55mV
= A10*423/1024 - 278
 * the temperature is transmitted as an intege
 r where 32.1 = 321
 * hence 4230 instead of 423
 */
```

```
//temp = results[0];
//degC = ((temp - 673) * 4230) / 1024;
//if( (*tempOffset) != 0xFFFF )
//{
 // degC += (*tempOffset);
//}

//temp = results[1];
//volt = (temp*25)/512;

/* Package up the data */
//msg[0] = degC&0xFF;
//msg[1] = (degC>>8)&0xFF;
//msg[2] = volt;

/* Send it over serial port */
//transmitDataString(1, addr, rssi, msg );

BSP_TOGGLE_LED1();

/* Done with measurement, disable flag */
sSelfMeasureSem = 0;
```

```
 * No critical section

 */
if (sPeerFrameSem)
{
  uint8_t      msg[3], len, i;

  /* process all frames waiting */
  for (i=0; i<sNumCurrentPeers; ++i)
  {
    if (SMPL_SUCCESS==SMPL_Receive(sLID[i]))
    {
      ioctlRadioSiginfo_t sigInfo;

      processMessage(sLID[i], msg, len);

      sigInfo.lid = sLID[i];

      SMPL_Ioctl(IOCTL_OBJ_RADIO,
      (void *)&sigInfo);

      value = msg[2]*100+msg[1]*10+msg[0];

      //transmitData( i, sigInfo.sigInfo.rssi,
      (char*)msg );
```

```
            printing[4]='0' + msg[0];
            printing[3]='0' + msg[1];
            printing[2]='0' + msg[2];

            TXString(printing, sizeof printing );
            BSP_TOGGLE_LED2();

            BSP_ENTER_CRITICAL_SECTION(intState);
            sPeerFrameSem--;
            BSP_EXIT_CRITICAL_SECTION(intState);
        }
    }
}
if (BSP_BUTTON1())
{
    __delay_cycles(2000000);  /* debounce */
    changeChannel();
}
else
{
    checkChangeChannel();
}
BSP_ENTER_CRITICAL_SECTION(intState);
if (sBlinky)
{
```

```
        if (++sBlinky >= 0xF)
        {
          sBlinky = 1;
          BSP_TOGGLE_LED1();
          BSP_TOGGLE_LED2();
        }
      }
      BSP_EXIT_CRITICAL_SECTION(intState);
    }

}

/* Runs in ISR context. */
static uint8_t sCB(linkID_t lid)
{
  if (lid)
  {
    sPeerFrameSem++;
    sBlinky = 0;
  }
  else
  {
    sJoinSem++;
  }
```

```
  /* leave frame to be read by application. */
  return 0;
}

static void processMessage(linkID_t lid,
uint8_t *msg, uint8_t len)
{

  /* do something useful */
  if (len)
  {
    //sprintf(printing, "a %d",*msg);
    //sprintf(printing, "Second element %d",msg[1]);

    BSP_TOGGLE_LED1();
  }
  return;
}

static void changeChannel(void)
{
#ifdef FREQUENCY_AGILITY
  freqEntry_t freq;

  if (++sChannel >= NWK_FREQ_TBL_SIZE)
```

```
  {
    sChannel = 0;
  }
  freq.logicalChan = sChannel;
  SMPL_Ioctl(IOCTL_OBJ_FREQ, IOCTL_ACT_SET, &freq);
  BSP_TURN_OFF_LED1();
  BSP_TURN_OFF_LED2();
  sBlinky = 1;
#endif
  return;
}

/* implement auto-channel-change policy here... */
static void checkChangeChannel(void)
{
#ifdef FREQUENCY_AGILITY
  int8_t dbm, inARow = 0;

  uint8_t i;

  memset(sSample, 0x0, SSIZE);
  for (i=0; i<SSIZE; ++i)
  {
    /* quit if we need to service an app frame */
    if (sPeerFrameSem || sJoinSem)
```

```
      {
        return;
      }
      NWK_DELAY(1);
      SMPL_Ioctl
      (IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_RSSI,
      (void *)&dbm);
      sSample[i] = dbm;

      if (dbm > INTERFERNCE_THRESHOLD_DBM)
      {
        if (++inARow == IN_A_ROW)
        {
          changeChannel();
          break;
        }
      }
      else
      {
        inARow = 0;
      }
    }
#endif
    return;
}
```

```
/*-------------------------------------------------------
* ADC10 interrupt service routine
-------------------------------------------------------*/
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{
  __bic_SR_register_on_exit(CPUOFF);
  // Clear CPUOFF bit from 0(SR)
}

/*-----------------------------------------------
* Timer A0 interrupt service routine
-----------------------------------------------*/
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A (void)
{
  sSelfMeasureSem = 0;
}
```

APPENDIX C

MINIMAL RF INTEGRATION CODE

```
#ifndef MRFI_H
#define MRFI_H


#include "bsp.h"
#include "mrfi_defs.h"


/* ------------------------------------------------------------------
 *       Defines
 * ------------------------------------------------------------------
 */
#define MRFI_NUM_LOGICAL_CHANS
__mrfi_NUM_LOGICAL_CHANS__

#define MRFI_NUM_POWER_SETTINGS
__mrfi_NUM_POWER_SETTINGS__

/* return values for MRFI_Transmit */
```

```
#define MRFI_TX_RESULT_SUCCESS          0
#define MRFI_TX_RESULT_FAILED           1

/* transmit type parameter for MRFI_Transmit */
#define MRFI_TX_TYPE_FORCED             0
#define MRFI_TX_TYPE_CCA                1


#ifndef SMPL_SECURE
#define  NWK_HDR_SIZE    3
#define  NWK_PAYLOAD     MAX_NWK_PAYLOAD
#else
#define  NWK_HDR_SIZE    6
#define  NWK_PAYLOAD     (MAX_NWK_PAYLOAD+4)
#endif

/* if external code has defined a maximum payload,
use that instead of default */
#ifdef MAX_APP_PAYLOAD
#ifndef MAX_NWK_PAYLOAD
#error ERROR: MAX_NWK_PAYLOAD not defined
#endif
#if MAX_APP_PAYLOAD < NWK_PAYLOAD
#define MAX_PAYLOAD  NWK_PAYLOAD
#else
```

```
#define MAX_PAYLOAD  MAX_APP_PAYLOAD
#endif
#define MRFI_MAX_PAYLOAD_SIZE
(MAX_PAYLOAD+NWK_HDR_SIZE)
/* SimpliciTI payload size plus six byte overhead */
#endif


/* frame definitions */
#define MRFI_ADDR_SIZE
__mrfi_ADDR_SIZE__
#ifndef MRFI_MAX_PAYLOAD_SIZE
#define MRFI_MAX_PAYLOAD_SIZE
__mrfi_MAX_PAYLOAD_SIZE__
#endif
#define MRFI_MAX_FRAME_SIZE
(MRFI_MAX_PAYLOAD_SIZE + __mrfi_FRAME_OVERHEAD_SIZE__)
#define MRFI_RX_METRICS_SIZE
__mrfi_RX_METRICS_SIZE__
#define MRFI_RX_METRICS_RSSI_OFS
__mrfi_RX_METRICS_RSSI_OFS__
#define MRFI_RX_METRICS_CRC_LQI_OFS
__mrfi_RX_METRICS_CRC_LQI_OFS__

/* Radio States */
```

```
#define MRFI_RADIO_STATE_UNKNOWN   0
#define MRFI_RADIO_STATE_OFF       1
#define MRFI_RADIO_STATE_IDLE      2
#define MRFI_RADIO_STATE_RX        3

/* Platform constant used to calculate worst-case for
an application
 * acknowledgment delay. Used in the NWK_REPLY_DELAY().
 *

#define    PLATFORM_FACTOR_CONSTANT
(2 + 2*(MAX_HOPS*(MRFI_CCA_RETRIES*
(8*MRFI_BACKOFF_PERIOD_USECS)/1000)))


/* ------------------------------------------------------
 *       Macros
 * ------------------------------------------------------
 */
#define MRFI_GET_PAYLOAD_LEN(p)
__mrfi_GET_PAYLOAD_LEN__(p)
#define MRFI_SET_PAYLOAD_LEN(p,x)
__mrfi_SET_PAYLOAD_LEN__(p,x)

#define MRFI_P_DST_ADDR(p)
```

```
__mrfi_P_DST_ADDR__(p)
#define MRFI_P_SRC_ADDR(p)
__mrfi_P_SRC_ADDR__(p)
#define MRFI_P_PAYLOAD(p)
__mrfi_P_PAYLOAD__(p)

/* ------------------------------------------------------
 *          Typdefs
 * ------------------------------------------------------
 */
typedef struct
{
  uint8_t frame[MRFI_MAX_FRAME_SIZE];
  uint8_t rxMetrics[MRFI_RX_METRICS_SIZE];
} mrfiPacket_t;


/* ------------------------------------------------------
 *          Prototypes
 * ------------------------------------------------------
 */
void    MRFI_Init(void);
uint8_t MRFI_Transmit(mrfiPacket_t *, uint8_t);
void    MRFI_Receive(mrfiPacket_t *);
void    MRFI_RxCompleteISR(void);
```

```
/* populated by code using MRFI */
uint8_t MRFI_GetRadioState(void);
void    MRFI_RxOn(void);
void    MRFI_RxIdle(void);
int8_t  MRFI_Rssi(void);
void    MRFI_SetLogicalChannel(uint8_t);
uint8_t MRFI_SetRxAddrFilter(uint8_t *);
void    MRFI_EnableRxAddrFilter(void);
void    MRFI_DisableRxAddrFilter(void);
void    MRFI_Sleep(void);
void    MRFI_WakeUp(void);
uint8_t MRFI_RandomByte(void);
void    MRFI_DelayMs(uint16_t);
void    MRFI_ReplyDelay(void);
void    MRFI_PostKillSem(void);
void    MRFI_SetRFPwr(uint8_t);

/* ------------------------------------------------
 *     Global Constants
 * ------------------------------------------------
 */
extern const uint8_t mrfiBroadcastAddr[];



 */
```

```
#endif
```